**Docker and Containerization of Software Programs**

The main challenge in software development arises when sharing the codebase with different stakeholders, such as developers and testers. To ensure that the code functions identically on everyone's local machine, it's crucial to replicate the exact same dependencies, libraries, and environment. This process can be time-consuming and prone to errors. This gives developers the ability to create predictable environments that are isolated from the rest of the applications and can be run anywhere.

Docker addresses this issue by offering containerization, which encapsulates the following three components into a single, portable unit:

1. Dependencies
2. Libraries
3. Code

A container functions similarly to a virtual machine (VM), but with key differences. While VMs require more hardware resources and have their own operating systems, making them heavy and resource-intensive, containers are lightweight. They create isolated environments on the host machine that utilize the same underlying hardware, ensuring efficient and consistent program execution. Containers provide most of the isolation of virtual machines at a fraction of the computing power.

Multiple containers can run simultaneously on a single machine without placing a heavy burden on system resources. To manage and orchestrate these containers, tools like Kubernetes are used. Kubernetes handles tasks such as load balancing and resource management in real-time, ensuring that each container has the necessary system resources based on its specific use case and execution requirements.

The key benefit of Docker is that it allows users to **package an application with all of its dependencies into a standardized unit** for software development.

Some of the main features of docker:

1. portability

2. more granular control over your resources giving your infrastructure improved efficiency which can result in better utilization of your compute resources.( Docker achieves granular control over resources by using containerization, which allows each container to be allocated specific amounts of CPU, memory, and storage. This precise allocation ensures that resources are used efficiently, improving overall system performance and resource utilization.)

This way, you can happily close your terminal and keep the container running. This is called **detached** mode.

$ docker run -d -P --name static-site prakhar1989/static-site

An important distinction to be aware of when it comes to images is the difference between base and child images.

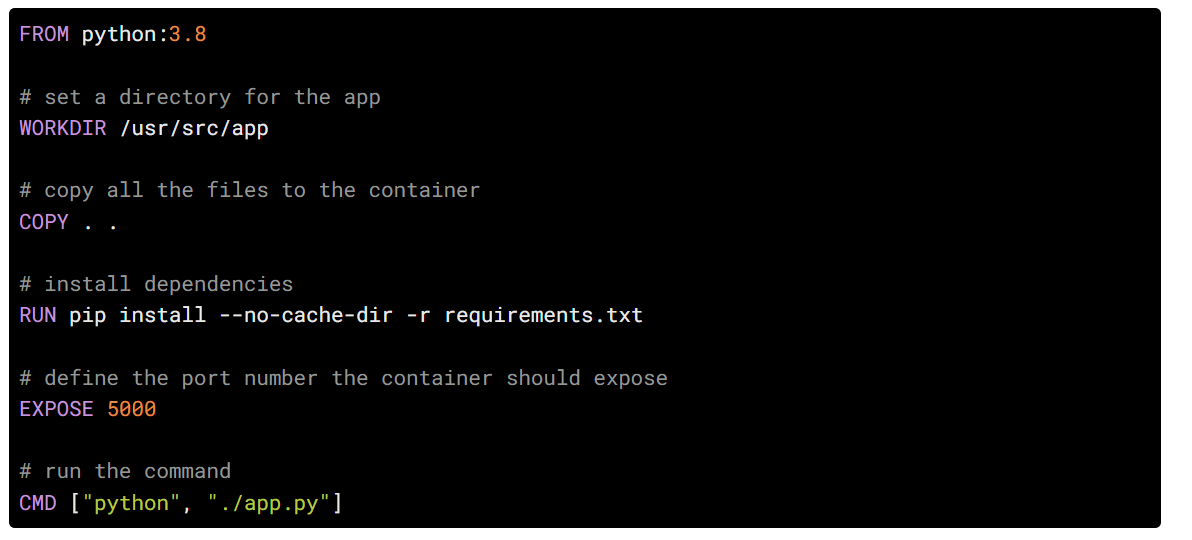
* **Base images** are images that have no parent image, usually images with an OS like ubuntu, busybox or debian.
* **Child images** are images that build on base images and add additional functionality.

Then there are official and user images, which can be both base and child images.

* **Official images** are images that are officially maintained and supported by the folks at Docker. These are typically one word long. In the list of images above, the python, ubuntu, busybox and hello-world images are official images.
* **User images** are images created and shared by users like you and me. They build on base images and add additional functionality. Typically, these are formatted as user/image-name.

A [Dockerfile](https://docs.docker.com/engine/reference/builder/) is a simple text file that contains a list of commands that the Docker client calls while creating an image

The docker build command does the heavy-lifting of creating a Docker image from a Dockerfile.



Now that we have our Dockerfile, we can build our image. The docker build command does the heavy-lifting of creating a Docker image from a Dockerfile.

docker run -p 8888:5000 yourusername/catnip

The command we just ran used port 5000 for the server inside the container and exposed this externally on port 8888. Head over to the URL with port 8888, where your app should be live.